Attribute grammars

- Attribute grammars act as an extension of context free grammars
- Each token and each nonterminal in the grammar has an associated set of attributes
- For each CFG rule in our grammar, we add a set of attribute rules: these identify two things
    - the combinations of valid attributes for the tokens on the RHS of the rule
    - how to compute the resulting attributes for the nonterminal on the LHS of the rule

Inherited vs synthesized attributes

- Considering the parse tree for a particular derivation, and looking at any given token or non-terminal:
  - its inherited attributes are those designated top down, from its parent/ancestors in the parse tree
  - its synthesized attributes are those computed bottom up from its children in the parse tree
- We've seen an ad hoc formulation of the attribute grammar rules in our use of yacc for context-sensitive checking

When are attribute rules applied?

- Dynamic methods:
  - assuming our compilation approach is based on the computation of a parse tree with attributes, this approach assumes we apply the attribute rules for a non-terminal after all its child nodes have been evaluated
- Rule-based methods:
  - encode a specific sequence to apply the various attribute rules, e.g. Once child #1 has been evaluated, apply rule #2 to each of the other children in sequence, then once they are all complete apply rule #3 to ... etc etc ...

Complexity of the rules

- Application of the rules in an automated fashion can become complex/slow
  - Consider a system with complex types and structural equivalence, where A,B,C are composite types
  - A := B + C might mean deep type structural type checking on B/C, then again on the result and A
- multipass systems (e.g. for declare anywhere), or the use of forward declarations can also lead to complex cross-referencing across portions of the annotated parse tree

Cyclic attribute grammars

- care must be taken to avoid developing attribute grammars with cyclic/mutually recursive dependencies, or the compiler can be locked in a loop trying to process the rules
- alternatively, some form of cycle-breaking evaluation method can be used to prevent the compiler from getting stuck, even if a cyclic set of dependencies is encountered

Parse tree availability and storage

- Implicitly or explicitly, these techniques rely on the generation of the parse tree to annotate: storing at least the subtrees needed to evaluate the current set of attribute rules
- For a large complex program, this can result in substantial storage needs

Autogenerated vs ad hoc

- You may have noticed that lex/yacc autogenerate the scanner and parser for us, but expect the compiler writer to create their own code to carry out context-sensitive checking
- Very challenging to implement both an effective attribute grammar and an efficient compiler-generator that applies the grammar
- The antlr/bison/yacc approach is widely used: automate the scanning and parsing, leave the rest for the devs